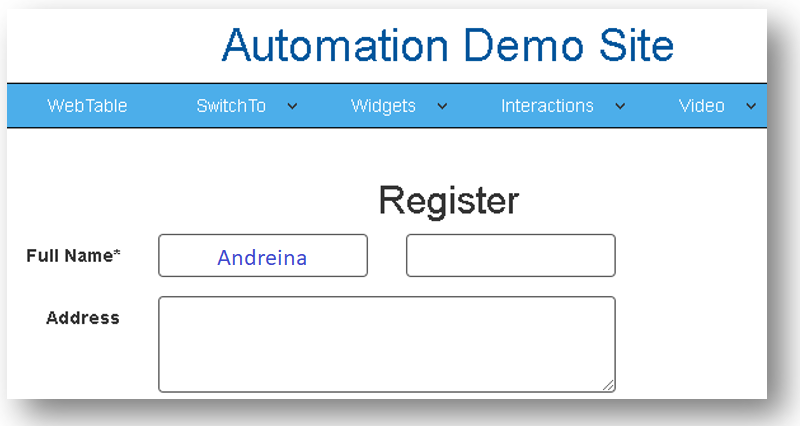
**¿Tip - Cómo crear “*Exceptions*” personalizadas?**

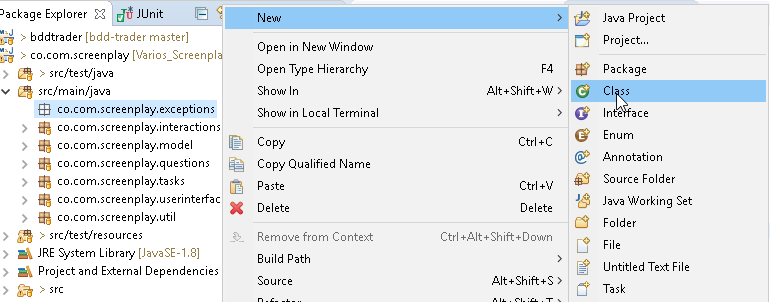
En muchos casos nuestras pruebas fallan debido a que se dan comportamientos inesperados para la naturaleza de la misma. Cuando esto sucede, Serenity, Selenium o el mismo Java maneja estos fallos o errores a través de las   
Exceptions. Entre las Exceptions que más se evidencian en nuestras pruebas tenemos “***NullPointerException***”, “***WebElementNotFoundException***” o “***AssertionError***”. En este documento encontrarás un tip para manejar de manera personalizada las Exceptions que se encuentren cuando necesitemos hacer **verificaciones.**

Para el ejemplo usaremos nuestro Reto 1 de Screenplay (La página de AutomationDemo) y supondremos que debemos verificar que al entrar a la página, el campo “First Name” debería venir con un nombre por defecto, digamos por ejemplo “Andreina”.

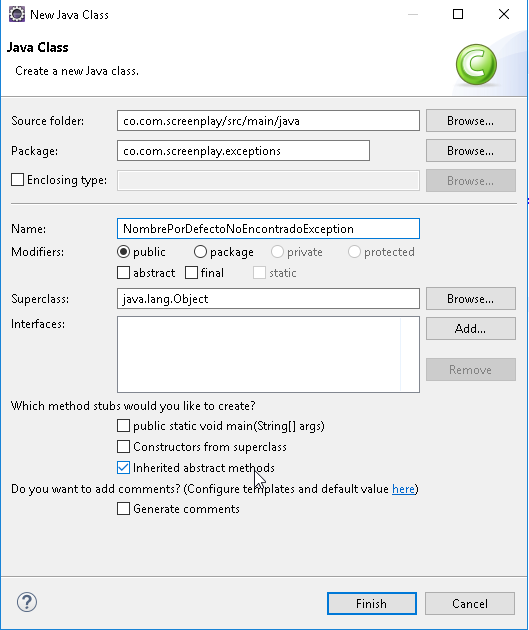


Si al entrar a esta página vemos que el elemento “First name” no tiene por defecto el nombre “Andreina” entonces debemos activar una Exception, porque algo no está saliendo como debería.

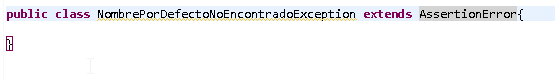
Para hacer nuestras Exceptions personalizadas entonces iremos al paquete “exceptions” en nuestro proyecto (src/main/java). (En caso de que no exista, entonces debemos crear un paquete con el nombre “exceptions”).



Le asignaremos un nombre que nos indique el tipo de Exception que manejaremos, para este ejemplo podría ser “NombrePorDefectoNoEncontradoException”

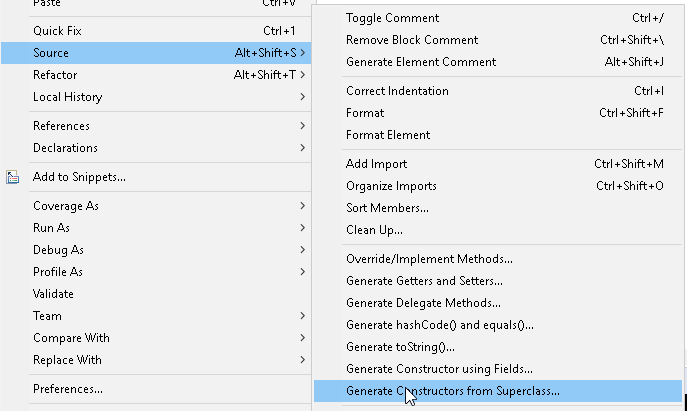


Como la exception que crearemos será activada debido a un error en la verificación, entonces haremos una herencia de la clase   
“***AssertionError***”, usando la palabra extends y luego el nombre de la clase que heredaremos, en este caso “***AssertionError***”.

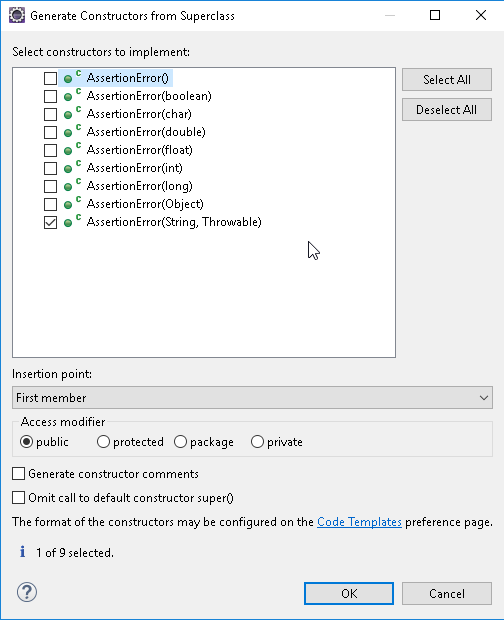


Cabe resaltar en este punto, que todas las Exceptions que tengan que ver con verificaciones las usaremos heredando de la clase ***AssertionError***.

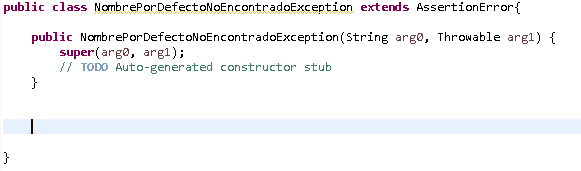
A esta nueva clase “***NombrePorDefectoNoEncontrado***” le crearemos un método constructor. Podemos hacer clic Derecho sobre la clase, ir a Source y luego a Generate Constructor from Superclass.

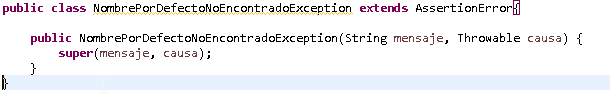


Y escogemos la opción que recibe por parámetro un String y un Throwable y le damos OK.

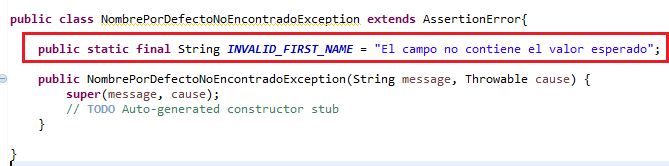


Inmediatamente se crea nuestro constructor con la siguiente estructura.

 Podemos cambiar el nombre de las variables para que sean algo más dicientes, y obtendríamos un resultado final como el que se muestra a continuación.



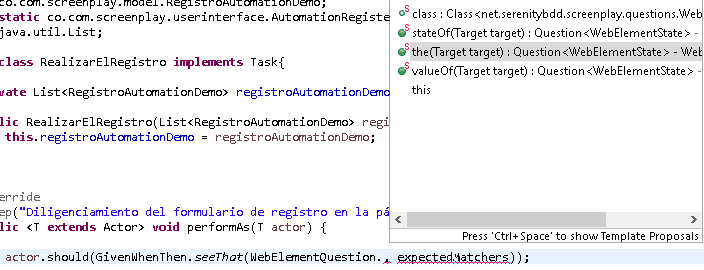
De igual manera dentro de la misma clase podemos definir cuáles serán esos mensajes personalizados que deseamos se muestren acorde al comportamiento que se presente, es decir, para nuestro caso vamos a definir un mensaje que se presentara cuando al verificar el campo “First name” no encuentre el valor por defecto que debe cargarse.



Ya hemos creado nuestra Exception personalizada, ahora solo nos resta usarla donde sea pertinente. Para esto iremos a nuestra tarea (task) donde diligenciamos el formulario y añadimos la verificación antes de escribir el nombre en el campo “First name”. Recuerda que las verificaciones son las sentencias que usamos normalmente en nuestro “Entonces” (@Then), por esto, usaremos los mismos métodos que usamos normalmente para ello.



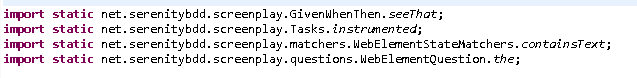
Y en el primer parámetro usaremos la clase ***WebElementQuestion*** y al presionar “.”, escogeremos la opción “***the***()”.



El método “***the***” pide un Target como parámetro, ahí escribiremos el Target donde se supone que estamos esperando que esté nuestro texto por Defecto, es decir, “Andreina”. Para el segundo parámetro, usaremos la clase “***WebElementStateMatchers***” y al escribir “.” Encontraremos una variedad de opciones, para este ejemplo usaremos el método llamado “***containsText***” y le enviaremos el String que debería aparecer en dicho **Target**. La instrucción nos quedaría de la siguiente forma.

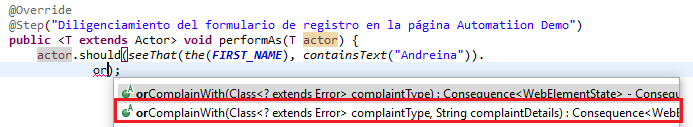


Para tener un código más legible, pondremos las clases “GivenWhenThen”, “WebElementStateMatchers” y “WebElementStateMatchers” como importaciones static y nos quedaría un código como el siguiente.





A esta sentencia solo le agregaremos al final el método “***orComplainWith***”, que nos ayudará en caso de que la verificación arroje un resultado negativo, es decir, que en el elemento no está escrito el texto esperado, entonces “arrojará” la Exception personalizada que hemos creado anteriormente.



Quedaria Asi:

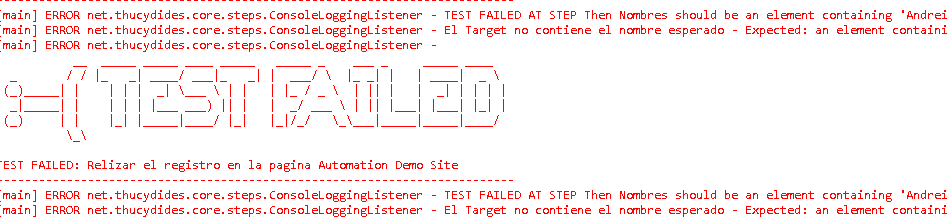


“***orComplainWith***” pide dos parámetros en el primer parámetro le enviaremos nuestra clase Exception personalizada y en el segundo parámetro un mensaje que nos indique por qué se generó la Exception, debe ser un mensaje que indique claramente la razón, en este caso seria uno de los mensajes que definimos en la clase “***NombrePorDefectoNoEncontradoException***”.

Al agregar estos parámetros obtendremos una sentencia como esta:



Al correr nuestra prueba encontraremos que fallará porque no encontrará el texto esperado, pero además el error nos indicará exactamente por qué se generó, y de esta forma saber dónde hacer la corrección.



Esta sentencia de verificación podemos usarla en cualquier lugar donde se requiera una verificación, en un “Task” o en el @Then del StepDefinitions, o incluso en una “Question”.

**¡Ahora inténtalo tú!**

**Webgrafía (Más detalle….)**

<http://thucydides.info/docs/articles/semantic-exceptions.html>